OSProj6 Banker's Algorithm

by 潘禧辰，518021910497

Menu

Abstract

- 编写了一个Banker's Algorithm simulator，完成了Banker's Algorithm的运行模拟
- 要求支持`*`、`RQ`、`RL`三种指令，分别完成打印available, maximum, allocation, need arrays的值、请求资源，释放资源

Environment

- Ubuntu 18.04
- Linux 5.3.0-42-generic
- VMware Workstation Rro 15.5.0 build-14665864

Quick Start

编译

Banker's Algorithm是用户态代码，直接使用如下gcc命令进行编译。

```
gcc bankers_algorithm.c -o bankers_algorithm
```

测试代码

maximum数组使用project提供的样例，存储在`maximum.txt`中

```
6,4,7,3
4,2,3,2
2,5,3,3
6,3,3,2
5,6,7,5
```

测试能否完成`RQ`和`RL`

```
./bankers_algorithm 9 9 9 9
> *
> RQ 0 5 4 3 2
> *
> RL 0 1 1 1 1
> *
```

测试能否在allocate资源足够的时候完成consumer

```
./bankers_algorithm 9 9 9 9
> *
> RQ 0 6 4 7 3
> *
```

测试能否判断unsafe state

```
./bankers_algorithm 8 8 8 8
> *
> RQ 0 3 3 3 3
> RQ 1 2 2 2 2
> RQ 2 2 2 2 2
> *
```

Implementation & Result

数据结构

定义了如下数据结构：

- `available`数组，用于存储每种resource的available amount
- `maximum`数组，用于存储每个customer对每种resource的maximum demand
- `allocation`数组，用于存储每个customer目前分配到的每种resource的数量
- `need`数组，用于存储每个customer对每种resource的remaining need数量
- `cmd`数组，用于读取第一个参数`argv[0]`
- `finished`数组，用于记录每个customer是否已经完成

```
/* the available amount of each resource */
int available[NUMBER_OF_RESOURCES];

/* the maximum demand of each customer_num */
int maximum[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];

/* the amount currently allocated to each customer_num */
int allocation[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];
```

```
/* the remaining need of each customer_num */
int need[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];

char cmd[5];
bool finished[NUMBER_OF_CUSTOMERS];
```

main

先完成了初始化和`maximum.txt`文件中`maximum`数组的读取了，再循环读取指令遇到`exit`指令退出循环结束程序，遇到`*`指令调用`show`函数打印available, maximum, allocation, need arrays的值，遇到`RQ`指令调用`request_resources`请求资源，遇到`RL`指令调用`release_resources`释放资源

```
int main(int argc, char *argv[]) {
    assert(argc == 5);
    for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
        available[i] = atoi(argv[i + 1]);
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
        finished[i] = false;
    FILE *f;
    f = fopen("maximum.txt", "r");

    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
            fscanf(f, "%d", &maximum[i][j]);
            fgetc(f);
        }

    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
            allocation[i][j] = 0;
            need[i][j] = maximum[i][j];
        }

    int array[NUMBER_OF_RESOURCES];
    while (true) {
        printf("> ");
        scanf("%s", cmd);

        if (strcmp(cmd, "exit") == 0)
            break;
        if (strcmp(cmd, "*") == 0) {
            show();
            continue;
        }
        if ((strcmp(cmd, "RQ") != 0) && (strcmp(cmd, "RL") != 0))
        {
            printf("ERROR\n");
            continue;
        }
        int customer_num;
        int request[NUMBER_OF_RESOURCES];
        scanf(" %d", &customer_num);
        for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
            scanf(" %d", &request[i]);
        if (strcmp(cmd, "RQ") == 0) {
            if (request_resources(customer_num, request) == -1)
```

```
                printf("request denied.\n");
            else
                printf("request accepted.\n");
        }
        else if (strcmp(cmd, "RL") == 0) {
            if (release_resources(customer_num, request) == -1)
                printf("release denied.\n");
            else
                printf("resource released.\n");
        }
    }
    fclose(f);
    return 0;
}
```

request_resources

请求资源时先判断指令是否正确，如果正确的话分别制作`finished_copy`，`available_copy`，`allocation_copy`，`need_copy`四个数组，将指令执行后`finished`，`available`，`allocation`，`need`的值存入其中，调用`check`判断指令执行后是否处于safe state，如果是的话就执行指令，修改`finished`，`available`，`allocation`，`need`。如果不是的话就不做修改

这里的`flag1`用于标识是否此次request过后，allocate资源足够的时候完成consumer。

`flag2`用于标识是是否处于safe state

```
int request_resources(int customer_num, int request[]) {
    if (customer_num > NUMBER_OF_CUSTOMERS - 1) {
        printf("error customer id.\n");
        return -1;
    }
    if (finished[customer_num] == true)
    {
        printf("customer already finished.\n");
        return -1;
    }
    for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
        if (available[i] < request[i]) {
            printf("resource not enough!\n");
            return -1;
        }

    bool flag1 = true;
    bool flag2;

    bool finished_copy[NUMBER_OF_CUSTOMERS];
    int available_copy[NUMBER_OF_RESOURCES];
    int allocation_copy[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];
    int need_copy[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];


    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        finished_copy[i] = finished[i];
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
            allocation_copy[i][j] = allocation[i][j];
            need_copy[i][j] = need[i][j];
        }
```

```
    }

    for (int i = 0; i < NUMBER_OF_RESOURCES; i++) {
        available_copy[i] = available[i] - request[i];
        allocation_copy[customer_num][i] += request[i];
        need_copy[customer_num][i] -= request[i];
    }

    for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
        if (need_copy[customer_num][i] != 0)
        {
            flag1 = false;
            break;
        }
    if (flag1)
    {
        finished_copy[customer_num] = true;
        for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
            available_copy[i] += allocation_copy[customer_num][i];
    }

    flag2 = check(finished_copy, available_copy, allocation_copy, need_copy);
    if (flag2) {
        for (int i = 0; i < NUMBER_OF_RESOURCES; i++) {
            available[i] -= request[i];
            allocation[customer_num][i] += request[i];
            need[customer_num][i] -= request[i];
        }
        if (flag1)
            finished[customer_num] = true;
        printf("safe state\n");
        return 0;
    }
    else
    {
        printf("unsafe state\n");
        return -1;
    }
}
```

release_resources

先判断指令是否合法，如果合法那么将指令执行后`available`，`allocation`，`need`的值进行修改

```
int release_resources(int customer_num, int release[]) {
    if (customer_num > NUMBER_OF_CUSTOMERS - 1) {
        printf("error customer id.\n");
        return -1;
    }
    for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
        if (allocation[customer_num][i] < release[i]) {
            printf("release too much!\n");
            return -1;
        }
    for (int i = 0; i < NUMBER_OF_RESOURCES; i++) {
        available[i] += release[i];
        allocation[customer_num][i] -= release[i];
```

```
        need[customer_num][i] += release[i];
    }
    return 0;
}
```

check

定义了三个bool变量：

- `flag`：表示第`i`个customer在目前资源下可否完成
- `non_finish`：表示是否完成了所有customer，如果是则为`false`，不是则为`true`
- `solve`：表示目前资源下是否有customr可被完成，如果没有则一定为unsafe state

`check`函数整体使用递归思路，递归结束条件是完成了所有customer`non_finish`为`false`，或者目前资源下没有customr可被完成`solve`为`false`

每次调用`check`函数都会选择一个可被完成的customer，将完成之后的finish, available, allocation, need arrays的值存入`Fin`，`Avail`，`Alloc`，`Need`并传入`check`检查该customer完成后是否是safe state。

```
bool check(bool Fin[], int Avail[], int Alloc[][NUMBER_OF_RESOURCES],int Need[]
[NUMBER_OF_RESOURCES]) {
    bool flag = false;
    bool non_finish = false;
    bool solve = false;
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
        if (!Fin[i])
        {
            non_finish = true;
            break;
        }
    if (!non_finish)
        return true;
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        if (Fin[i])
            continue;
        flag = false;
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++)
            if (Avail[j] < Need[i][j]) {
                flag = true;
                break;
            }
        if (flag)
            continue;

        solve = true;
        Fin[i] = true;
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++)
            Avail[j] += Alloc[i][j];
        break;
    }
    if (!solve)
        return false;
    if (check(Fin, Avail, Alloc, Need))
        return true;
    else
        return false;
```

```
}
```

show

循环打印available, maximum, allocation, need arrays的值

```
void show() {
    printf("Available:\n");
    for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
        printf("Resource%d: %d\t", i, available[i]);
    printf("\n");

    printf("Maximum:\n");
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        printf("Customer%d:\t", i);
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
            printf("Resource%d: %d\t", j, maximum[i][j]);
        }
        printf("\n");
    }

    printf("Allocation:\n");
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        if (finished[i] == true)
            continue;
        printf("Customer%d:\t", i);
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
            printf("Resource%d: %d\t", j, allocation[i][j]);
        }
        printf("\n");
    }

    printf("Need:\n");
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        if (finished[i] == true)
            continue;
        printf("Customer%d:\t", i);
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
            printf("Resource%d: %d\t", j, need[i][j]);
        }
        printf("\n");
    }
}
```

完整代码

将上述几个部分组合得到：

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <assert.h>

#define NUMBER_OF_CUSTOMERS 5
#define NUMBER_OF_RESOURCES 4
```

```
/* the available amount of each resource */
int available[NUMBER_OF_RESOURCES];

/* the maximum demand of each customer_num */
int maximum[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];

/* the amount currently allocated to each customer_num */
int allocation[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];

/* the remaining need of each customer_num */
int need[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];

int request_resources(int customer_num, int request[]);

int release_resources(int customer_num, int release[]);

bool check(bool Fin[], int Avail[], int Alloc[][NUMBER_OF_RESOURCES],int Need[]
[NUMBER_OF_RESOURCES]);

void show();

char cmd[5];
bool all_finished = false;
bool finished[NUMBER_OF_CUSTOMERS];

int main(int argc, char *argv[]) {
    assert(argc == 5);
    for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
        available[i] = atoi(argv[i + 1]);
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
        finished[i] = false;
    FILE *f;
    f = fopen("maximum.txt", "r");

    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
            fscanf(f, "%d", &maximum[i][j]);
            fgetc(f);
        }

    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
            allocation[i][j] = 0;
            need[i][j] = maximum[i][j];
        }

    int array[NUMBER_OF_RESOURCES];
    while (true) {
        printf("> ");
        scanf("%s", cmd);

        if (strcmp(cmd, "exit") == 0)
            break;
        if (strcmp(cmd, "*") == 0) {
            show();
            continue;
        }
```

```
        int customer_num;
        int request[NUMBER_OF_RESOURCES];
        scanf(" %d", &customer_num);
        for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
            scanf(" %d", &request[i]);
        if (strcmp(cmd, "RQ") == 0) {
            if (request_resources(customer_num, request) == -1)
                printf("request denied.\n");
            else
                printf("request accepted.\n");
        }
        else if (strcmp(cmd, "RL") == 0) {
            if (release_resources(customer_num, request) == -1)
                printf("release denied.\n");
            else
                printf("resource released.\n");
        } else
            printf("ERROR\n");
    }
    return 0;
}

int request_resources(int customer_num, int request[]) {
    if (customer_num > NUMBER_OF_CUSTOMERS - 1) {
        printf("error customer id.\n");
        return -1;
    }
    if (finished[customer_num] == true)
    {
        printf("customer already finished.\n");
        return -1;
    }
    for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
        if (available[i] < request[i]) {
            printf("resource not enough!\n");
            return -1;
        }

    bool flag1 = true;
    bool flag2;

    bool finished_copy[NUMBER_OF_CUSTOMERS];
    int available_copy[NUMBER_OF_RESOURCES];
    int allocation_copy[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];
    int need_copy[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];


    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        finished_copy[i] = finished[i];
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
            allocation_copy[i][j] = allocation[i][j];
            need_copy[i][j] = need[i][j];
        }
    }

    for (int i = 0; i < NUMBER_OF_RESOURCES; i++) {
        available_copy[i] = available[i] - request[i];
        allocation_copy[customer_num][i] += request[i];
```

```
        need_copy[customer_num][i] -= request[i];
    }

    for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
        if (need_copy[customer_num][i] != 0)
        {
            flag1 = false;
            break;
        }
    if (flag1)
    {
        finished_copy[customer_num] = true;
        for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
            available_copy[i] += allocation_copy[customer_num][i];
    }

    flag2 = check(finished_copy, available_copy, allocation_copy, need_copy);
    if (flag2) {
        for (int i = 0; i < NUMBER_OF_RESOURCES; i++) {
            available[i] -= request[i];
            allocation[customer_num][i] += request[i];
            need[customer_num][i] -= request[i];
        }
        if (flag1)
            finished[customer_num] = true;
        printf("safe state\n");
        return 0;
    }
    else
    {
        printf("unsafe state\n");
        return -1;
    }
}

int release_resources(int customer_num, int release[]) {
    if (customer_num > NUMBER_OF_CUSTOMERS - 1) {
        printf("error customer id.\n");
        return -1;
    }
    for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
        if (allocation[customer_num][i] < release[i]) {
            printf("release too much!\n");
            return -1;
        }
    for (int i = 0; i < NUMBER_OF_RESOURCES; i++) {
        available[i] += release[i];
        allocation[customer_num][i] -= release[i];
        need[customer_num][i] += release[i];
    }
    return 0;
}

bool check(bool Fin[], int Avail[], int Alloc[][NUMBER_OF_RESOURCES],int Need[]
[NUMBER_OF_RESOURCES]) {
    bool flag = false;
    bool non_finish = false;
    bool solve = false;
```

```
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++)
        if (!Fin[i])
        {
            non_finish = true;
            break;
        }
    if (!non_finish)
        return true;
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        if (Fin[i])
            continue;
        flag = false;
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++)
            if (Avail[j] < Need[i][j]) {
                flag = true;
                break;
            }
        if (flag)
            continue;

        solve = true;
        Fin[i] = true;
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++)
            Avail[j] += Alloc[i][j];
        break;
    }
    if (!solve)
        return false;
    if (check(Fin, Avail, Alloc, Need))
        return true;
    else
        return false;
}

void show() {
    printf("Available:\n");
    for (int i = 0; i < NUMBER_OF_RESOURCES; i++)
        printf("Resource%d: %d\t", i, available[i]);
    printf("\n");

    printf("Maximum:\n");
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        printf("Customer%d:\t", i);
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
            printf("Resource%d: %d\t", j, maximum[i][j]);
        }
        printf("\n");
    }

    printf("Allocation:\n");
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        if (finished[i] == true)
            continue;
        printf("Customer%d:\t", i);
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
            printf("Resource%d: %d\t", j, allocation[i][j]);
        }
        printf("\n");
```

```
    }

    printf("Need:\n");
    for (int i = 0; i < NUMBER_OF_CUSTOMERS; i++) {
        if (finished[i] == true)
            continue;
        printf("Customer%d:\t", i);
        for (int j = 0; j < NUMBER_OF_RESOURCES; j++) {
            printf("Resource%d: %d\t", j, need[i][j]);
        }
        printf("\n");
    }
}
```

结果

- 测试能否完成`RQ`和`RL`

```
pan@pan-virtual-machine:~/桌面/osproj/6$ ./bankers_algorithm 9 9 9 9
> *
Available:
Resource0: 9    Resource1: 9    Resource2: 9    Resource3: 9
Maximum:
Customer0:      Resource0: 6    Resource1: 4    Resource2: 7    Resource3: 3
Customer1:      Resource0: 4    Resource1: 2    Resource2: 3    Resource3: 2
Customer2:      Resource0: 2    Resource1: 5    Resource2: 3    Resource3: 3
Customer3:      Resource0: 6    Resource1: 3    Resource2: 3    Resource3: 2
Customer4:      Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 5
Allocation:
Customer0:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer1:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer2:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer3:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer4:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Need:
Customer0:      Resource0: 6    Resource1: 4    Resource2: 7    Resource3: 3
Customer1:      Resource0: 4    Resource1: 2    Resource2: 3    Resource3: 2
Customer2:      Resource0: 2    Resource1: 5    Resource2: 3    Resource3: 3
Customer3:      Resource0: 6    Resource1: 3    Resource2: 3    Resource3: 2
Customer4:      Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 5
> RQ 0 5 4 3 2
safe state
request accepted.
> *
Available:
Resource0: 4    Resource1: 5    Resource2: 6    Resource3: 7
Maximum:
Customer0:      Resource0: 6    Resource1: 4    Resource2: 7    Resource3: 3
Customer1:      Resource0: 4    Resource1: 2    Resource2: 3    Resource3: 2
Customer2:      Resource0: 2    Resource1: 5    Resource2: 3    Resource3: 3
Customer3:      Resource0: 6    Resource1: 3    Resource2: 3    Resource3: 2
Customer4:      Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 5
Allocation:
Customer0:      Resource0: 5    Resource1: 4    Resource2: 3    Resource3: 2
Customer1:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer2:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer3:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer4:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Need:
Customer0:      Resource0: 1    Resource1: 0    Resource2: 4    Resource3: 1
Customer1:      Resource0: 4    Resource1: 2    Resource2: 3    Resource3: 2
Customer2:      Resource0: 2    Resource1: 5    Resource2: 3    Resource3: 3
Customer3:      Resource0: 6    Resource1: 3    Resource2: 3    Resource3: 2
Customer4:      Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 5
```

```
> *
Available:
Resource0: 4    Resource1: 5    Resource2: 6    Resource3: 7
Maximum:
Customer0:      Resource0: 6    Resource1: 4    Resource2: 7    Resource3: 3
Customer1:      Resource0: 4    Resource1: 2    Resource2: 3    Resource3: 2
Customer2:      Resource0: 2    Resource1: 5    Resource2: 3    Resource3: 3
Customer3:      Resource0: 6    Resource1: 3    Resource2: 3    Resource3: 2
Customer4:      Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 5
Allocation:
Customer0:      Resource0: 5    Resource1: 4    Resource2: 3    Resource3: 2
Customer1:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer2:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer3:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer4:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Need:
Customer0:      Resource0: 1    Resource1: 0    Resource2: 4    Resource3: 1
Customer1:      Resource0: 4    Resource1: 2    Resource2: 3    Resource3: 2
Customer2:      Resource0: 2    Resource1: 5    Resource2: 3    Resource3: 3
Customer3:      Resource0: 6    Resource1: 3    Resource2: 3    Resource3: 2
Customer4:      Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 5
> RL 0 1 1 1 1
resource released.
> *
Available:
Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 8
Maximum:
Customer0:      Resource0: 6    Resource1: 4    Resource2: 7    Resource3: 3
Customer1:      Resource0: 4    Resource1: 2    Resource2: 3    Resource3: 2
Customer2:      Resource0: 2    Resource1: 5    Resource2: 3    Resource3: 3
Customer3:      Resource0: 6    Resource1: 3    Resource2: 3    Resource3: 2
Customer4:      Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 5
Allocation:
Customer0:      Resource0: 4    Resource1: 3    Resource2: 2    Resource3: 1
Customer1:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer2:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer3:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer4:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Need:
Customer0:      Resource0: 2    Resource1: 1    Resource2: 5    Resource3: 2
Customer1:      Resource0: 4    Resource1: 2    Resource2: 3    Resource3: 2
Customer2:      Resource0: 2    Resource1: 5    Resource2: 3    Resource3: 3
Customer3:      Resource0: 6    Resource1: 3    Resource2: 3    Resource3: 2
Customer4:      Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 5
```

- 测试能否在allocate资源足够的时候完成consumer

```
pan@pan-virtual-machine:~/桌面/osproj/6$ gcc bankers_algorithm.c -o bankers_algorithm
pan@pan-virtual-machine:~/桌面/osproj/6$ ./bankers_algorithm 9 9 9 9
> *
Available:
Resource0: 9    Resource1: 9    Resource2: 9    Resource3: 9
Maximum:
Customer0:      Resource0: 6    Resource1: 4    Resource2: 7    Resource3: 3
Customer1:      Resource0: 4    Resource1: 2    Resource2: 3    Resource3: 2
Customer2:      Resource0: 2    Resource1: 5    Resource2: 3    Resource3: 3
Customer3:      Resource0: 6    Resource1: 3    Resource2: 3    Resource3: 2
Customer4:      Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 5
Allocation:
Customer0:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer1:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer2:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer3:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer4:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Need:
Customer0:      Resource0: 6    Resource1: 4    Resource2: 7    Resource3: 3
Customer1:      Resource0: 4    Resource1: 2    Resource2: 3    Resource3: 2
Customer2:      Resource0: 2    Resource1: 5    Resource2: 3    Resource3: 3
Customer3:      Resource0: 6    Resource1: 3    Resource2: 3    Resource3: 2
Customer4:      Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 5
> RQ 0 6 4 7 3
safe state
request accepted.
> *
Available:
Resource0: 3    Resource1: 5    Resource2: 2    Resource3: 6
Maximum:
Customer0:      Resource0: 6    Resource1: 4    Resource2: 7    Resource3: 3
Customer1:      Resource0: 4    Resource1: 2    Resource2: 3    Resource3: 2
Customer2:      Resource0: 2    Resource1: 5    Resource2: 3    Resource3: 3
Customer3:      Resource0: 6    Resource1: 3    Resource2: 3    Resource3: 2
Customer4:      Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 5
Allocation:
Customer1:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer2:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer3:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer4:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Need:
Customer1:      Resource0: 4    Resource1: 2    Resource2: 3    Resource3: 2
Customer2:      Resource0: 2    Resource1: 5    Resource2: 3    Resource3: 3
Customer3:      Resource0: 6    Resource1: 3    Resource2: 3    Resource3: 2
Customer4:      Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 5
>
```

- 测试能否判断unsafe state

```
pan@pan-virtual-machine:~/桌面/osproj/6$ ./bankers_algorithm 8 8 8 8
> *
Available:
Resource0: 8    Resource1: 8    Resource2: 8    Resource3: 8
Maximum:
Customer0:      Resource0: 6    Resource1: 4    Resource2: 7    Resource3: 3
Customer1:      Resource0: 4    Resource1: 2    Resource2: 3    Resource3: 2
Customer2:      Resource0: 2    Resource1: 5    Resource2: 3    Resource3: 3
Customer3:      Resource0: 6    Resource1: 3    Resource2: 3    Resource3: 2
Customer4:      Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 5
Allocation:
Customer0:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer1:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer2:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer3:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer4:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Need:
Customer0:      Resource0: 6    Resource1: 4    Resource2: 7    Resource3: 3
Customer1:      Resource0: 4    Resource1: 2    Resource2: 3    Resource3: 2
Customer2:      Resource0: 2    Resource1: 5    Resource2: 3    Resource3: 3
Customer3:      Resource0: 6    Resource1: 3    Resource2: 3    Resource3: 2
Customer4:      Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 5
> RQ 0 3 3 3 3
safe state
request accepted.
> RQ 1 2 2 2 2
safe state
request accepted.
> RQ 2 2 2 2 2
unsafe state
request denied.
> *
Available:
Resource0: 3    Resource1: 3    Resource2: 3    Resource3: 3
Maximum:
Customer0:      Resource0: 6    Resource1: 4    Resource2: 7    Resource3: 3
Customer1:      Resource0: 4    Resource1: 2    Resource2: 3    Resource3: 2
Customer2:      Resource0: 2    Resource1: 5    Resource2: 3    Resource3: 3
Customer3:      Resource0: 6    Resource1: 3    Resource2: 3    Resource3: 2
Customer4:      Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 5
Allocation:
Customer0:      Resource0: 3    Resource1: 3    Resource2: 3    Resource3: 3
Customer1:      Resource0: 2    Resource1: 2    Resource2: 2    Resource3: 2
Customer2:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer3:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Customer4:      Resource0: 0    Resource1: 0    Resource2: 0    Resource3: 0
Need:
Customer0:      Resource0: 3    Resource1: 1    Resource2: 4    Resource3: 0
Customer1:      Resource0: 2    Resource1: 0    Resource2: 1    Resource3: 0
Customer2:      Resource0: 2    Resource1: 5    Resource2: 3    Resource3: 3
Customer3:      Resource0: 6    Resource1: 3    Resource2: 3    Resource3: 2
Customer4:      Resource0: 5    Resource1: 6    Resource2: 7    Resource3: 5
>
```

Difficulties

- 一开始忽略了c语言中数组传递时并不是值传递，导致在`check`函数中对几个数组做出的修改影响了全局变量数组。修改写法后问题解决

Reference

- Operating System Concept 10^{th} edition
- Source code for the 10th edition of Operating System Concepts https://github.com/greggagne/osc10e